High Assurance Software Engineering (HASE) Plan for NGSCB/Unity

TCAAB Review
August 5 2004

Brandon Baker – Pen Test/Red Team Lead
Tim Oerting – Test Manager

Agenda

- HASE Goals & Overview
- Questions for TCAAB
- Look at Individual HASE Processes
- Q&A

Unity HASE Goals

- Provide High Assurance for Windows. Bridge the divide between standard Microsoft development practices and gov't high security efforts
- Well understood and verifiable behavior of code with fewer vulnerabilities esp. in critical components.
- Reduction of attack surfaces and containment of flaws through well specified, layered design
- Reduce the number of exploits to meet the needs of security sensitive enterprises (ie: Financial, Medical, Government)
- Clear evidence of the level of assurance of the system, documented non-goals and calculated acceptable risks isolated from core components.
- Provide a foundation for future efforts to achieve greater assurance and security certification (ie: MLS for Military/Intel)
- Credibility among experts who will have developed independent levels of confidence in the assurance of the system.

HASE Team

Team is cross discipline to support high assurance across varied team roles:

- **Core Members:**
  - Brandon Baker – Pen Test Lead
  - Christine Chew – Program Management Lead
  - Simon McMahon – HASE Program Manager
  - Marcus Peinado – Architect
  - Mike Marr – High Assurance Development
  - Tim Oerting – Test Manager
- **Key Supporting Members:**
  - Ken Ray – Development Manager
  - Mark Schreiffer – Build/Lab Engineer
  - Matt Stipes – Group Program Manager
  - Paul England – Architecture Lead
  - Suyash Sinha – Test Lead
  - Tony Ureche – Program Manager

What is HASE and How Did We Get Here?

HASE = High Assurance Software Engineering

- Combined effort of processes / tools / techniques to assure that the security principles of our system our upheld, that our design is robust, and to prevent breaks of the system.
- Reviewed past industry experience and took best practices
  - Took input from TCSEC, CC, local experts (Lipner, Aucsmith, etc..), other design process paradigms (originally targeted EAL6)
  - Reviewed published reports from design/verification of previous high security / high assurance systems. Kept the most important concepts to be in our plan.
- Extended the new Microsoft Security Development Lifecycle (SDL) to add Higher Assurance specifics

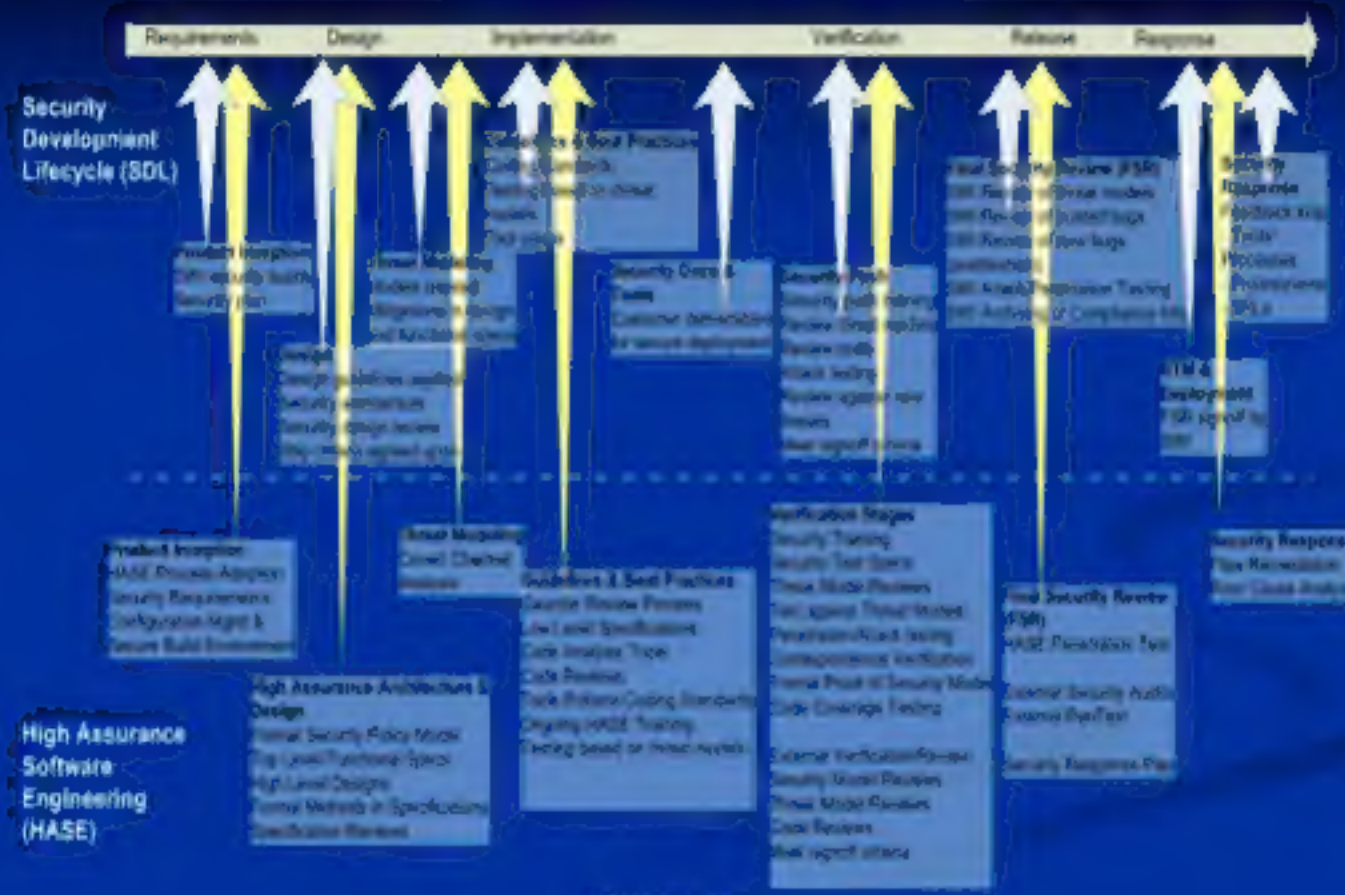
HASE Formation for Unity

- We took a set of elements/concepts that we felt layer well together to build HASE.
  - No one element by itself provides high assurance. It is the combination of all elements that does.
  - Each element may have varying levels of implementation
    - Determine a 'sweet spot' on a per component basis
    - Driven by the Unity TCB, Security Requirements, etc.
  - Formed a cross discipline virtual team to focus on driving these concepts into the product.
  - Rollout of these elements is ongoing
  - HASE is an organic thing. We are refining as we go to do the best job for the product and for you all.
    - It is not a one size fits all process and we are tailoring the level of rigor / assurance requirements to the individual components.

Questions for TCAAB

- We would like to get feedback on our HASE process for implementing High Assurance at Microsoft.
  - Does our process seem sufficient to achieve our high assurance goals?
  - What additions/changes/refinements would you suggest we investigate?
  - Can you provide specific assistance or review on some of these areas?

HASE and the SDL



HASE Elements

- **Design/Infrastructure**
  - Security Requirements
  - Design Documentation
  - Layering, Modularization, TCB Minimization
  - Threat & Privacy Modeling Analysis
  - Secure Build Environment
  - Formal Specification of Critical System Components
  - Unity Team HASE Training
- **Implementation**
  - Coding Standards & Annotations
  - Mandatory code review process
  - Correspondence evaluation to tie implementation to spec's & requirements
- **Verification**
  - Testing from formal & informal specs, Threat Models
  - Red Team / Penetration Test Team
  - Gated staging to mainline verification process ('Gauntlet')
  - HASE Tools
- **Release / Post-Release**
  - Secure delivery/install
  - Update/Patching plan
  - Common Criteria certification (EAL 4+)

Security Requirements and Defining the TCB

- Defines the system
- Sets prioritization for project decisions
- Tells us how what we build should operate
- Constrains feature creep
- Provides justification for security vs. {performance | size | complexity | wiz-bang nifty must have feature} decisions
- Helps support layered design
- Guides HASE focus
  - Limited resources
  - HASE can be expensive, invest the most in critical areas first
- Helps identify critical assets and attack surface
  - Where will we be attacked?
  - What will attackers want to steal?
  - Defines what a "compromise" for the system means

Security and Assurance Requirements

Design Documentation
and Configuration Management

- **Goals**
  - Precisely define goals and designs
  - Refine designs at each level
  - Support correspondence analysis
  - Ensure team buy-off & understanding
  - Provide auditable documentation
  - Support post-ship maintenance of system
- **Document Types**
  - Requirements Documents
  - Top Level Functional Spec
  - High Level Design Documents
  - Low Level Design Documents
  - Process Documents
- **Review Tool** – submit review in Source Depot
  - Reviewers assigned by HASE team
  - Reviews at different levels

Layering, Factoring & Minimization

- Goal: lower the number and severity of bugs in the TCB through good design
  - Identify security assets (HW & SW) and their accessors / modifiers TSF
  - Separate Non-TSF functions from TSF functions where possible & Break up modules where appropriate
  - Run modules at the least privilege possible
  - Avoid unnecessary interactions between modules
  - Minimize cross-dependencies between layers
  - Minimize the complexity of the entire TSF
  - Design access control and/or information flow control policies so they are simple enough to be analyzed
  - Omit irrelevant functions from the TSF

Threat Modeling

- Guides the design by forcing designer analyze the system in terms of
  - Assets
  - Threats
  - Exposed interfaces
  - Data flow
- Follow data as it comes in externally (potentially malicious) and analyze what functions act on it
- Follow assets as they travel through the system and analyze what functions act on them – do any expose the data?
- Identify points in which externally supplied data and critical assets are processed by the same function – exposed to exploits
- Capture the thought process that occurs when designing security, but in a formalized, repeatable way
- Helps validate all possibilities for compromise
- Mitigations identified early can be well integrated into the design and implementation
- Focuses attention to critical areas; highlights non-goals
- Security features (mitigations) without a corresponding threat are a waste of time and resources
- Considering covert channels

Formal Specification and Analysis

- Forces team to think about the design in a rigorous way
- Goals:
  - Remove ambiguity in design details
  - Model code to ensure design is correct
  - Invariants can be shown to hold on model
    - Correspondence is weak with informal specifications
  - Find bugs that standard ways of testing miss
  - Ensure we know what we were building, while we were building it, ensure what we build is what we specified

Application of Formal Methods

- Historical Efforts:
  - Specified 2 critical NGSCB components using ASML and TLA:
    - PTEC (Page Table Edit Control) and SRM (Security Reference Monitor)
  - Training in TLA provided to team to attempt to have devs write formal specs
    - Difficulties with skill set and acceptance
- Current Plans:
  - Contract with Formal Methods shop to assist in application and training
  - Specify Hypervisor external interfaces and optionally other critical components (ie: memory isolation ≙ shadow page tables; security policy)

Code Reviews

- Goals
  - Consistently encourage & enforce best coding practices and guidelines
  - Validate the implementation matches intended design and upholds security requirements of the system
  - Catch bugs at the earliest, most inexpensive stage possible
- Types of Reviews
  - Internal: Informal & Formal
  - External: 3rd party security experts
- What's Involved
  - Developer self-evaluation
  - Reviewer verification & analysis
  - Systematic tool verification & analysis
  - Correspondence verification

Correspondence

- Ensure consistency of design and security principles at multiple levels of abstraction and that code matches those designs
  - Requirements are upheld by design
  - Lower level designs uphold higher level constraints
  - RS->TLFS->HLD->LLD->Code
- Multiple solutions
  - 1. Formal Methods and model checkers are able to show consistency between levels of abstraction/specification
  - 2. Traceability: Interfaces and security principles can be labeled and traced between abstractions.
    - 2.1 Generate Header files from specs (LLD). An enforced mapping of interfaces between spec and code.
  - 3. Testing: Test cases built at higher levels of abstraction should always hold for all lower levels

Testing

- Testing in general is an integral part of achieving high assurance
- Verification against specifications
  - Correspondence of Test cases to spec to cover functionality, edge cases, error cases, layering, performance goals
  - Test cases built from formal models where possible
- Specific test cases for threat model/security requirements to ensure mitigations work
- Code Coverage helps ensure tests are executing all interfaces and decision branches
  - Potential to also validate coverage against formal models

Penetration Testing

- Dedicated Full-Time Team
- Penetration testing attempts to force the system outside of it's boundaries
  - Pound on exposed interfaces
  - Validates assumptions
  - Simulates future attacker
- 3rd party security audits = fresh set of eyes to look at system without taint of assumptions
  - Specifically 3rd party security experts

Secure Build Environment and the Gauntlet Process

- Configuration Management
  - All files are tracked in DB with reviewers assigned
- Spec Review & Code Review process
  - All file signoffs stored in DB with verification against required reviewer list & verification of 3rd party reviewers (reviewer != submitter)
  - Smart Cards for signoff
- Physically Secure environment with split staging & approved trees to prevent tampering
  - Ensure built binaries are based on correct, reviewed files
  - Ensure change tracking and review signoffs are not tampered with
  - Build/Source machine access limited and two-key system
  - Limit ability of malicious users & code from impacting the environment (Private Domain/network, VPN/Firewall)
- Signoff / Migration to "Approved" tree
- Logging and Auditing - checkins, review signoffs, and DB data and schema changes

HASE Tools

- Tools to help with the Development and Verification Process
  - Leverage Prefix/Prefast and other MS best practice tools
  - Code and specification review tracking tools
  - Generation of code headers from specification to ensure code and spec correspondence
  - Layering & dependency constraint validation tools
  - Verification of adherence to coding standards
  - Annotation tools to improve code analysis

Updating/Patching

- **Goal**
  - Provide ongoing assurance after shipping
  - Resolve issues of resealing secrets without compromising security policy
- **Mechanisms**
  - Vehicle to verify and release critical updates
    - Leverage existing Windows Update infrastructure/tools
  - Response plan for handling critical issues
    - Coordination with MSRC

Rigor and Prioritization

- Identify for each component:
  - Security critical assets
  - Privilege levels
  - Security functions
  - Hardware security functions
- Break system into priorities based on the identified factors
- Map subcomponents to appropriate level of rigor based on priorities

Certification

- Current plan
  - Part of Windows Longhorn CC certification
    - EAL4+ against SLOS PP(Single Level OS Protection Profile)
    - CC efforts driven within STBU by Steve Lipner
- *Potential Opportunities*
  - *New Protection Profile in draft: Separation Kernels for High Robustness Environments*
    - *Appears to align well with parts of our hypervisor plans*
    - *Issues: draft is at high EAL6+, requires scheduler not in HV v1 plans*

Questions & Answers

- Past Experiences – What has been most successful, from your view, for achieving high assurance in software development?
- Does our process seem sufficient to achieve our high assurance goals?
  - What additions/changes/refinements would you suggest we investigate?
  - Can you provide specific assistance or review on some of these areas?